

SENIOR DESIGN PROJECT REPORT

Instrumentation and Data Collection for Sheet Glass Production

Submitted to

Professor Elaine Cooney
Electrical Engineering Technology Program
Engineering Technology Department

by
David Shaw

March 5, 2019

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 2 of 45
		Document No.:1	Version: 5.6

Abstract

Kokomo Opalescent Glass (KOG) is a manufacturer of art glass located in Kokomo Indiana. KOG has high defect rates in their sheet glass production process that can vary greatly depending on operator experience and environmental factors. This project aimed to improve the repeatability of KOG's sheet glass production process by enabling them to monitor the temperature at which glass sheets enter their annealing oven and to decrease their defect rate, which has historically been around 25%. Through integrating instrumentation and data collection into KOG's production process, defects in sheet glass production were successfully decreased by approximately 10% in the weeks following the installation of the device created in this project.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 3 of 45
		Document No.:1	Version: 5.6

Table of Contents

Abstract	2
Table of Contents	3
Revision History	4
Introduction	5
Problem Statement	5
System Overview	5
Referenced Documents	6
System-wide Design Decisions.....	7
Hardware.....	7
Software	8
Bill of Materials	9
Installation Images	10
Software	13
External Interfaces	18
Internal Interfaces	19
User Setup and Operation	20
Testing Plan and Results	21
Conclusions and Recommendations	22
Code Appendix	23
MainPage.xaml.cs	23
MAX7219_LED_Driver.cs.....	28
TextFileLogManager.cs	32
CTL_IR_Sensor_Diver.cs.....	32
VBA macro code module running in Excel	41
From VBA macro code library written by me	42
ADC_I2C.cs.....	43

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 4 of 45
		Document No.:1	Version: 5.6

Revision History

Version	Date	Revised by	Description
1.0	April 26, 2018	David Shaw	Initial version
2.0	July 19, 2018	David Shaw	Full draft version
3.0	July 29, 2018	David Shaw	Final Draft
4.0	August 03, 2018	David Shaw	Final Version
5.0	February 16, 2019	David Shaw	Final Version 2.0
5.5	February 21, 2019	David Shaw	Edits
5.6	February 27, 2019	David Shaw	Final Proofread

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 5 of 45
		Document No.:1	Version: 5.6

Introduction

This report outlines the design of the instrumentation, controls, and data collection systems created for KOG and explains the reasoning behind design decisions.

Problem Statement

Glass sheets are formed by rollers and must first harden before being pushed into an annealing oven to cool gradually. However, if a sheet enters the annealing oven too soon the glass will deform inside of the oven or fuse with rust and other debris, and if the glass is pushed in too late the sheet will likely crack or shatter inside of the oven due to internal stresses.

Prior to the installation of this system, KOG did not have an accurate or repeatable way to determine the correct moment to push the glass sheets into the oven and instead relied on temperature estimates made by workers based on glass color or when it “looks right.” These temperature estimates can differ greatly from one person to another based on their level of experience and skill.

System Overview

The system’s purpose is to accurately and non-intrusively measure the temperature of newly formed sheets of glass. The temperature of the glass is displayed in clear view of the operator so that they know the current temperature of the glass. The system counts the number of sheets as they’re produced and records two temperatures for each sheet: directly after being formed and right before it is pushed into the oven. All recorded information is periodically saved to a log file stored on the device and is accessible through a Wi-Fi connection to the device or via an Ethernet connection. An accompanying database was created and is kept on an office computer for importing and formatting the data from the log file.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 6 of 45
		Document No.:1	Version: 5.6

Referenced Documents

Table 1: Reference Documents

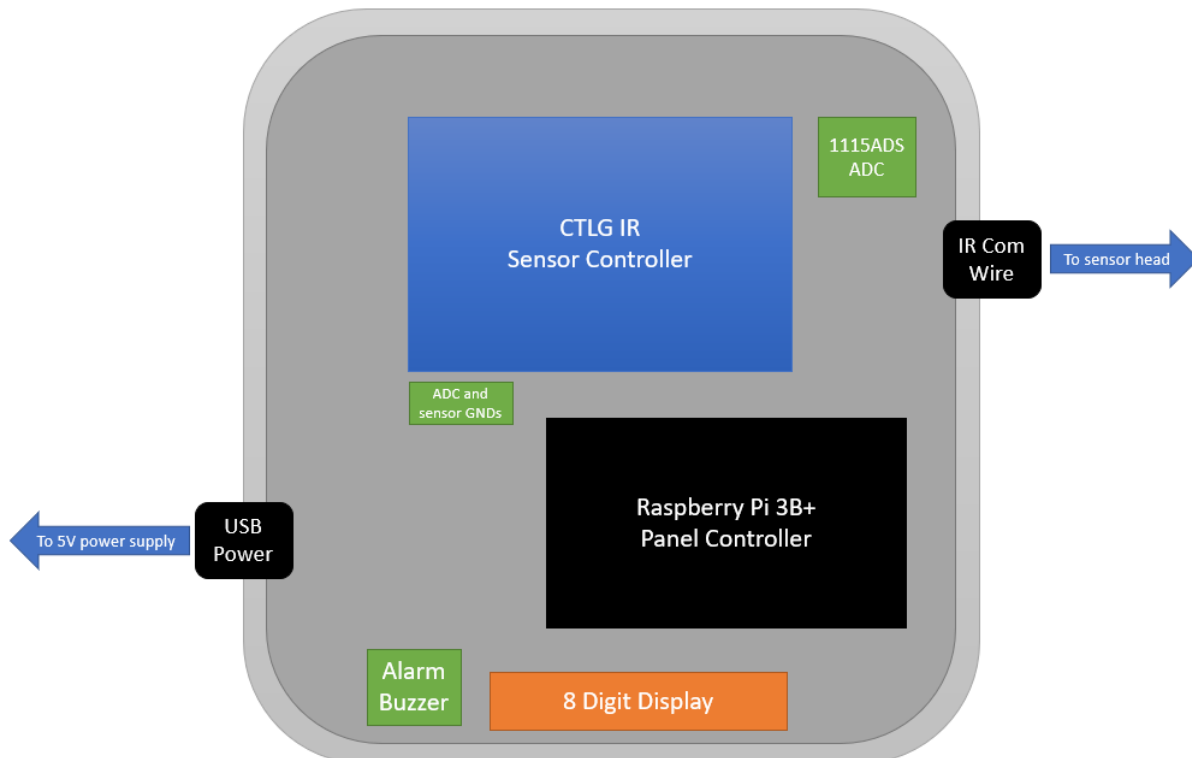
Title	Document Reference Number	Comment
Operating Instructions thermoMETER CTL	https://www.micro-epsilon.com/download/manuals/man--thermoMETER-CT--en.pdf	IR sensor Manual
MAX7219 - 8 Digit LED Display Module Driver for ESP8266	https://www.instructables.com/id/MAX7219-8-Digit-LED-Display-Module-Driver-for-ESP8/	Guide on creating a basic driver for the display
MAX7219/MAX7221 Serially Interfaced, 8-Digit LED Display Drivers	https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf	Display documentation
thermoMETER CT-CTL communication interface	https://www.micro-epsilon.com/download/manuals/man--thermoMETER-CT--en.pdf	IR sensor commands
ADS111x Ultra-Small, Low-Power, I2C- Compatible, 860-SPS, 16- Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator	http://www.ti.com/product/ADS1115/technicaldocuments	Documentation for ADS1115 ADC

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 7 of 45
		Document No.:1	Version: 5.6

System-wide Design Decisions

- Sensor – An IR sensor was selected as the method to obtain measurements because of its ability to safely and unobtrusively obtain measurements from a distance. Both of these qualities were desired because of the extreme temperatures of the glass and because of the need to not introduce additional defects into the glass. The type of IR sensor must also be selected on the basis of being able to accurately measure both clear and opaque glass.
- Display – Needs to be highly visible and low cost.
- Controller – Must be able to handle all of the necessary IO for peripherals, have Wi-Fi capability so files can be accessed remotely, be a system that I have some experience with, and be low cost.
- Control Panel – Must be able to hold all components, be low cost, be able to withstand a warm and dusty environment.
- ADC – Needed to be capable of measuring two analog pins inside the sensor controller, for ambient and object temperature. Must be low cost.

Hardware



Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 8 of 45
		Document No.:1	Version: 5.6

Major components and the reasoning behind their selection

- IR Sensor – The Micro-Epsilon CTLG-SF45L-C3 was chosen for its high temperature measurement range, built in laser sight, and ease of use. This sensor is specifically designed for measuring the temperature of hot sheet glass by looking at a narrower range of the infrared spectrum in the 5-micron range. IR sensors that observe a wider bandwidth of the infrared spectrum will give lower readings due to the inclusion of the radiation being reflected by the table under the glass.
- Display – The MAX7219 digital 8-digit display was selected for its low cost and high visibility. It is controlled through a four-wire serial interface.
- Controller – The Raspberry Pi 3 B was selected as the controller because of its large number of IO pins, built in Wi-Fi capability, built in USB ports, and relatively low cost.
- Control Panel – The PolyCase.com SK-28 SK Series Enclosures with Knockouts was selected because of its small size, low cost, clear front panel, and the high heat resistance and strength of its polycarbonate material.
- ADC – The ADS1115 ADC was chosen for its simple I2C interface, ability to measure up to four analog inputs, and low cost.

Software

- IR Sensor – The Micro-Epsilon CTLG-SF45L-C3 comes with multiple software programs for connecting to it with a PC via USB.
- Operating System – Windows 10 IoT was selected because of my prior experience with it and because I thought that it was a more complete version of windows that would make connecting to various peripheral devices easy. This ended up being a partially incorrect assumption because the core version of the operating system does not include all of the windows functionality for serial communication.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 9 of 45
		Document No.:1	Version: 5.6

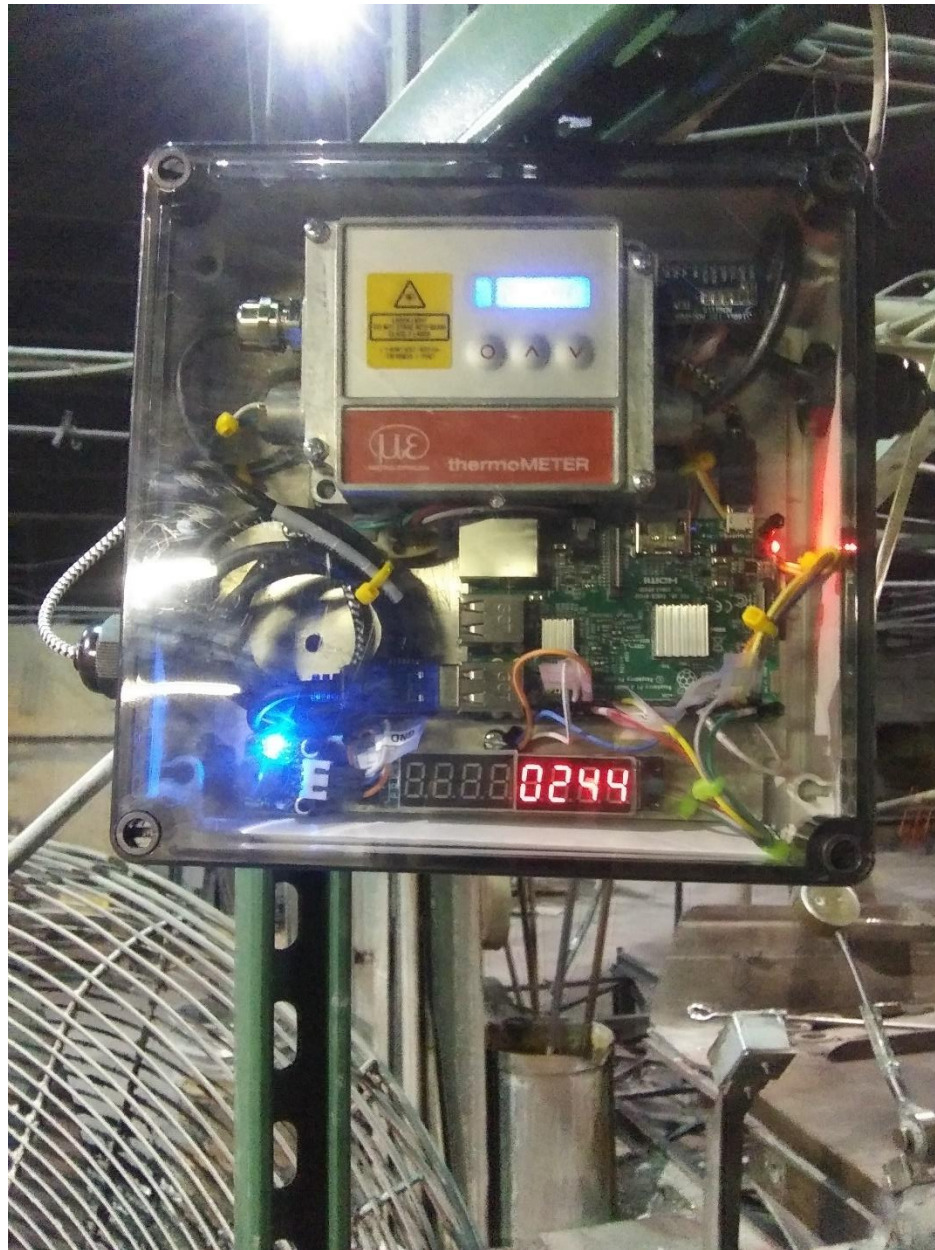
Bill of Materials

Qty.	Description	Vendor	Cost
1	Raspberry Pi 3B+ board	Amazon.com	\$48.99
1	SD card for Rasbery Pi	Amazon.com	\$9.50
1	5V .5A power supply for Pi	Amazon.com	Included in kit
1	Heatsinks for Rasbery Pi	Amazon.com	Included in kit
1	Clear covered polycarbonate electronics enclosure with knockouts, IP66 rating	Polycase.com	\$33.36
1	Fixed mount industrial infrared CTLG sensor with USB interface	Micro-Epsilon	\$1,100
1	IR sensor mounting brackets	Micro-Epsilon	\$35
1	C channel for sensor and controller mounting	Local hardware store	\$40
1	ADS1115 Analog to Digital Converter	Amazon.com	\$6.99
	Wires for internal connections	Amazon.com	
1	8-digit digital display module	Amazon.com	\$7.90
1	Internal mounting panel for poly case	Polycase.com	\$10.36
1	Active low buzzer for high temperature and error alarms	Amazon.com	\$5.59
1	Plastic PCB board standoffs	Amazon.com	\$8.99
1	Package of plastic water proof cable glands	Amazon.com	\$8.99
1	Zip ties for cable management	Amazon.com	\$4.99

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 10 of 45
		Document No.:1	Version: 5.6

Installation Images

The finished and mounted control panel in operation



Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 11 of 45
		Document No.:1	Version: 5.6

The IR sensor head mounted



Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 12 of 45
		Document No.:1	Version: 5.6

Complete system view

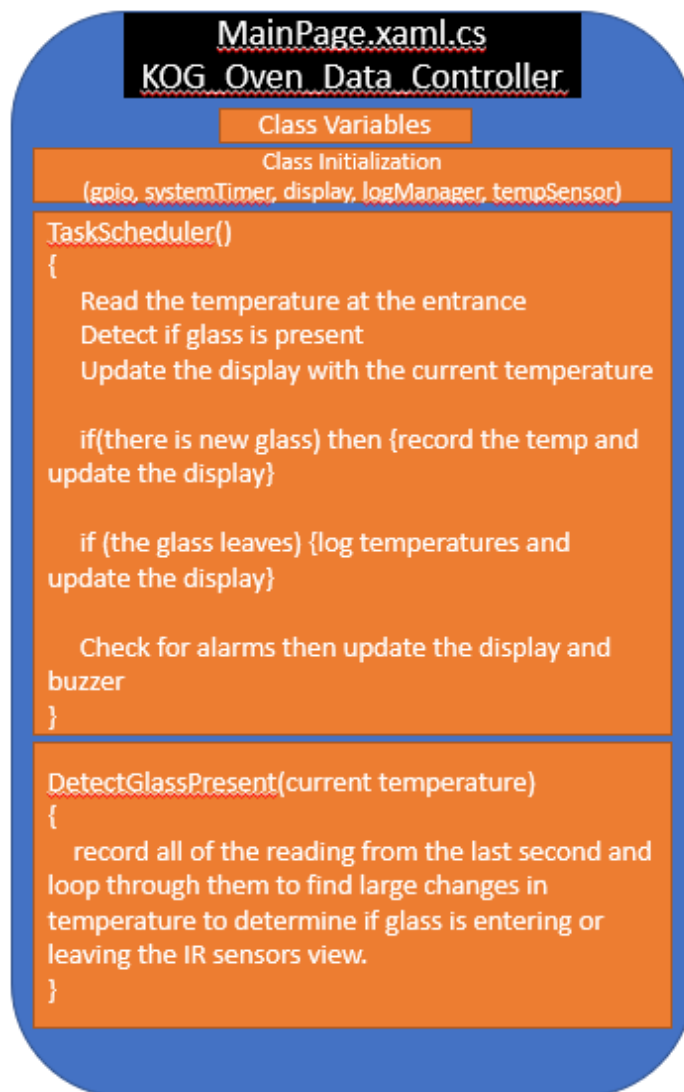


Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 13 of 45
		Document No.:1	Version: 5.6

Software

The controller was loaded with Windows 10 IoT Core and programmed using Visual Studio and C#. The universal windows program code consisted of four major classes.

- KOG_Oven_Data_Controller
 - Top level class which controls program flow and utilizes all other classes.
 - Code Appendix 1



Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 14 of 45
		Document No.:1	Version: 5.6

- MAX7219_LED_Driver.cs
 - Driver for controlling the MAX7219 display via serial communication.
 - Code Appendix 2

MAX7219_LED_Driver.cs

MAX7219_Driver

Class Variables and commands

Class Initialization
 Setup (gpio, brightness, scan limit, decode mode, test, zero)

WriteByte(data)
 {
 write one bit a time to the DIN pin then toggle the CLK pin
 }

SetRegister(register, value)
 {
 write one bit a time to the DIN pin then toggle the CS pin
 }

PrintNumber(number)
 {
 put each digit of the number into a separate variable then
 output each variable to its respective register
 }

ZeroAll()
 Use a loop and SetRegister() to zero the display

AdjustBrightness_0_To_8(level)
 Use SetRegister() to adjust the brightness of the display

DisplayTest()
 use PrintNumber() to loop from 0 to 9999

WarningBlink()
 use AdjustBrightness to blink between the minimum and
 maximum brightness values while display a number code to
 correspond to the warning or error

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 15 of 45
		Document No.:1	Version: 5.6

- TextFileLogManager.cs
 - Code for creating and writing formatted data to any number of text log files.
Three log files were used: DATA_LOG, SYSTEM_LOG, and ERROR_LOG
 - Code Appendix 3

TextFileLogManager.cs
TextFileLog

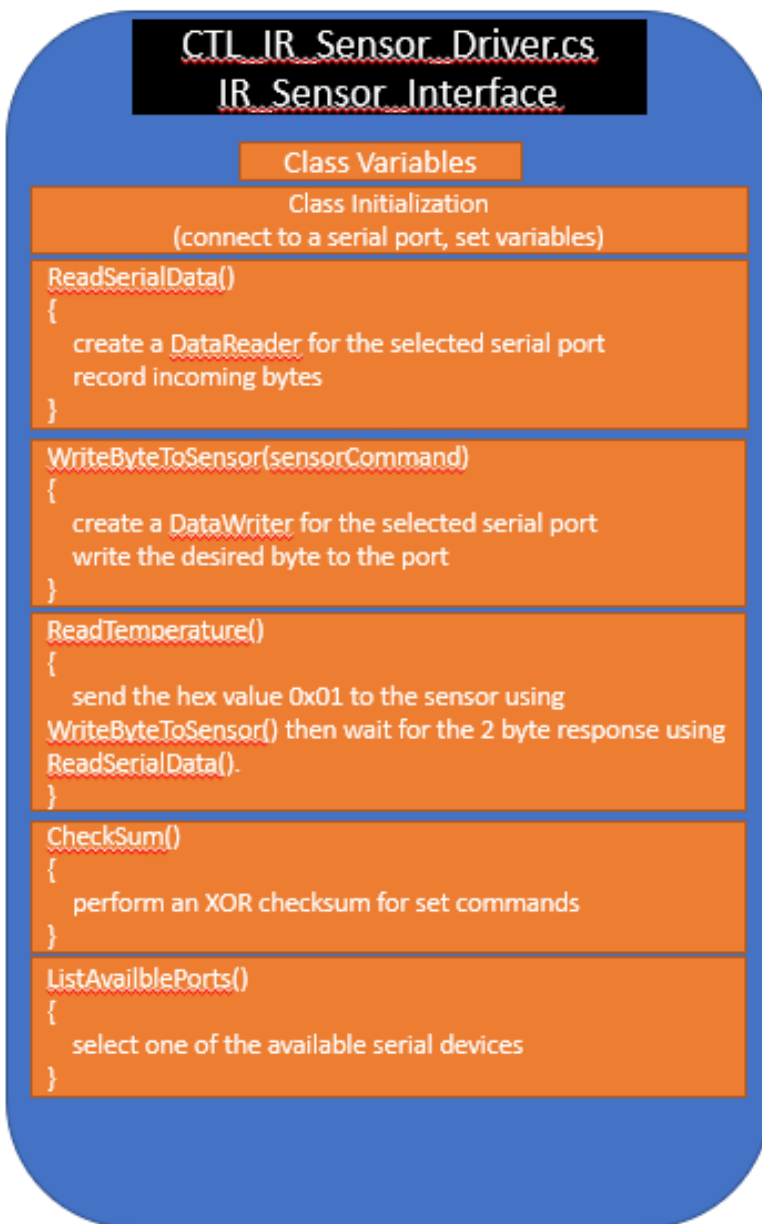
Class Variables

Class Initialization

AddLogEntry(logName, textEntry)
{
Find the apps local folder the find the desired file in that folder.
Open the file if it exists or create a new one if it doesn't.
Append the new text and the current date and time to the end of the file.
}

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 16 of 45
		Document No.:1	Version: 5.6

- CTL_IR_Sensor_Driver.cs (was not used in final version, replaced by ADC)
 - Commands and serial communication drivers for interfacing with the IR sensor over a USB cable.
 - Code Appendix 4



- CTL_IR_Sensor_Driver.cs (was not used in final version, replaced by ADC)

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 17 of 45
		Document No.:1	Version: 5.6

- Commands and serial communication drivers for interfacing with the IR sensor
- Code Appendix 7

ADC_I2C.cs

Read Analog Pins

Class Variables

Class Initialization

```

StartConversion(analogInput number)
{
    write a two byte command to a register to start a
    single-shot conversion of the analog voltage
}

```

```

InitializeAsync()
{
    Initialize the I2c communication between the
    Raspberry Pi and the ADC. Then write to the ADC to
    configure it before setting up an output pin to
    trigger when a voltage conversion is complete.
}

```

- File Transfer
 - File transfer is handled through the built in Windows 10 IoT interface.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 18 of 45
		Document No.:1	Version: 5.6

- When running on the same network, a web browser can access the Raspberry Pi controller and log files can be copied to the user's computer using an IP address or the link and login information shown below.

E	F
Controller Location	http://192.168.137.79:8080/#File%20explorer
Username	Administrator
Password	123

External Interfaces

The system's user interface consists of an 8-digit numeric display mounted inside the control panel and of an Excel spreadsheet that will be run on a PC and used as a tool for viewing the log files.

The Excel database is meant to be an output interface used by management or engineers to track and monitor production.

The IR sensor's controller also has a built-in three-button interface for changing sensor settings. This interface can be accessed by opening the front panel. IR sensor setting can also be changed via included software running on a laptop and connected to the IR sensor via USB. This interface does not need to be accessed during regular production and is only intended for use by engineers.

The device will interface with its environment through the temperature readings of the IR sensor.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 19 of 45
		Document No.:1	Version: 5.6

A screenshot of the excel database with some sample data collected during testing.

	A	B	C	D	E
1	Production Data			Update	
2	Date & Time ▼	Enter Temp °F ▼	Exit Temp °F ▼		
3					
4					
5	23/01/2019 6:51:23 PM	1021.82	1021.82		
6	23/01/2019 6:53:36 PM	1021.82	1021.82		
7	23/01/2019 6:55:30 PM	1021.82	1021.82		
8	23/01/2019 6:57:22 PM	1021.82	1021.82		
9	23/01/2019 6:57:28 PM	1021.82	1021.82		
10	23/01/2019 6:59:19 PM	1021.82	1021.82		
11	23/01/2019 7:00:54 PM	1021.82	1021.82		
12	23/01/2019 7:19:45 PM	1021.82	1021.82		
13	23/01/2019 7:21:31 PM	1021.82	1021.82		

See code in Appendix 6.

Internal Interfaces

Serial – The controller interfaces with both the display through serial communication. The display uses a four-pin method with a GND, DIN, CS, and CLK pins.

I2C – The controller connected to the ADS1115 ADC through a two pin I2C interface as well as through a 3.5V and GND pin.

USB – The USB connecting the sensor controller to the raspberry pi is only used to provide power to the sensor but can also be removed and connected to a PC for debugging and changing of sensor setting through the software provided with the sensor.

Analog – The ADS1115 ADC has two connections to the IR sensor. It's first channel is connected to an analog output for the object temperature and it's second channel is connected to the analog output for the ambient temperature inside the sensor's head. The GNDs for the ADC, IR sensor, and controller are also all tied to together.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 20 of 45
		Document No.:1	Version: 5.6

Wireless – The Raspberry Pi controller will be wirelessly connected to KOG’s network and can be accessed through the Windows 10 IoT interface using it’s IP address. A PC in the office can then log in to download the log files for viewing.

User Setup and Operation

The control panel is setup and coded so that it only needs to be turned on to operate with no further input necessary. The device can sometime take about a minute to fully startup.

To make changes to the IR sensor’s settings the user can open the front of the control panel and use the sensor controller’s three button interface to make changes, but this is not a part of regular operation.

A different set of users will also interact with an Excel workbook running on a PC. This Excel workbook is programmed with VBA scripts and has a button to automatically open and import data from the SheetGlassTemperatureLog.txt file and then format this data inside a table. See 4.3.1

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 21 of 45
		Document No.:1	Version: 5.6

Testing Plan and Results

After installation, the system was tested using the following plan.

- Temperature Reading of Non-glass Surfaces
 - **Plan** — Measuring only the temperature of the table conveyor at the entrance of the oven, take note of the number shown on the display. Compare this number to the value read on a hand-held IR sensor.
 - **Result** – The table temperature was accurately read and closely matched the temperature read by the hand-held IR sensor. The temperature of the table steadily increases throughout the day as more sheets of hot glass passed over it.
- Data Transfer
 - **Plan** – After starting up the panel, use a computer to access the log files stored on the Raspberry Pi via Wi-Fi. Ensure that both the controller and the PC are connected to the same local network.
 - **Result** – The files were found and downloaded to a PC both wirelessly and via Ethernet.
- Temperature Reading of Sheet Glass
 - **Plan** - Produce multiple sheets of glass and confirm that the display is showing a number in the expected range of molten glass. The handheld IR sensor should show a significantly lower temperature since it is reading a wider bandwidth of the IR spectrum and will detect the lower temperature of the table under the sheet. Produce multiple sheets of glass to check for consistency.
 - **Result** – The sensor consistently read a temperature in the 1300°F to 1200°F range as the glass entered its field of view, and the hand-held IR sensor always read a temperature a few hundred degrees lower. This difference shows that my selection of sensor was a good one because the surface below the glass was having a minimal impact on readings. The temperature of the glass will not always be the same as it

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 22 of 45
		Document No.:1	Version: 5.6

enters the IR sensors view because the amount of time the glass spends being mixed in the previous station differs from batch to batch.

- Process Improvement
 - **Plan** - Check for defects in the glass sheets at the exit of the oven and determine if the higher consistency of temperature at entering the oven has had a positive effect on the scrap rate.
 - **Result** – KOG used the device in their everyday production for two weeks and reported that their breakage was decreased by approximately 10%, increasing their average production rate of non-cracked and defect free glass sheets from around 75% to 85%.

Conclusions and Recommendations

My device performs all of the primary functions that I originally set out to perform with it, and test results have shown that my project was successful in significantly decreasing material and time loss for KOG by decreasing their sheet glass breakage rates by approximately 10%. The system can also be utilized in the integration of further sensors and other peripherals in the future.

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 23 of 45
		Document No.:1	Version: 5.6

Code Appendix

MainPage.xaml.cs

```

1. using System;
2. using Windows.UI.Xaml;
3. using Windows.UI.Xaml.Controls;
4. using Windows.Devices.Gpio;
5. using MAX7219_LED_Driver;
6. using TextFileLogManager;
7. using System.Diagnostics;//for debugging
8. using ADC;
9.
10. /// <summary>
11. ///
12. /// Control the temperature monitoring and logging system for the entrance to the
    annealing oven at KOG
13. /// David Shaw 2018
14. ///
15. /// </summary>
16.
17. namespace KOG_Oven_Data_Controller
18. {
19.     public sealed partial class MainPage : Page
20.     {
21.         //create an instance of the 8digit display driver
22.         private MAX7219_Driver display = new MAX7219_Driver();
23.         //create an instance of the text file manager class
24.         private TextFileLog logManager = new TextFileLog();
25.         //create instance of ADC for reading sensor temp
26.         private ADC_I2C ADC = new ADC_I2C();
27.
28.         private double currentObjectTemp = 0.0;//stores the latest value read by
the IR sensor
29.         private double currentAmbiantTemp = 0.0;//stores the latest value of the
IR sensor's ambient temp
30.         private double[] pastTemps = new double[4];//to store the last four
temperature readings for making comparisions (1 second worth of readings when
taken every 250ms)
31.         private double glassInitialTemp = 0.0;
32.         private bool glassPresent = false; //is glass currently being measured
33.         private bool prevGlassPresent = false;
34.         private bool newData = false;
35.         private int programCounter = 0;//used by the task scheduler
36.
37.         //pin assignments for an alarm buzzer
38.         private const int BUZZ = 26;//pin 37
39.         private GpioPin buzz;
40.         private GpioPinValue buzz_value;
41.

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 24 of 45
		Document No.:1	Version: 5.6

```

42.         //timers
43.         private DispatcherTimer systemTimer;
44.
45.         //Log file names
46.         private const string DATA_LOG = "SheetGlassTemperatureLog.txt";
47.         private const string SYSTEM_LOG = "SystemLog.txt";
48.         private const string ERROR_LOG = "ErrorLog.txt";
49.
50.         public MainPage()
51.         {
52.             this.InitializeComponent();
53.
54.             //creat a new instance of the gpio controller
55.             var gpio = GpioController.GetDefault();
56.
57.             //initialize the pins for the alarm buzzer
58.             //It is an active low buzzer so low=on and high=off
59.             buzz = gpio.OpenPin(BUZZ);
60.             buzz.SetDriveMode(GpioPinDriveMode.Output);
61.             buzz_value = GpioPinValue.High;
62.
63.             //initialize the timer for the task scheduler to be called every 250
ms
64.             systemTimer = new DispatcherTimer();
65.             systemTimer.Interval = TimeSpan.FromMilliseconds(250);// .25 second
delay
66.             systemTimer.Tick += TaskScheduler;
67.             systemTimer.Start();
68.
69.             //initialize the display
70.             display.InitDisplay();
71.             display.SetScanLimit_0_To_7(3);
72.             display.AdjustBrightness_0_To_8(0);
73.             //display.DisplayTest();
74.
75.             //initialize the log
76.             logManager.InitializeLog();
77.             logManager.AddLogEntry(SYSTEM_LOG, "Power ON");
78.
79.             ADC.InitializeAsync();
80.         }
81.
82.         //control program flow and timing
83.         //called every 250ms
84.         public void TaskScheduler(object sender, object e)
85.         {
86.             programCounter++;
87.
88.             //the first time each function is called it requests a temperature
reading

```


Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 25 of 45
		Document No.:1	Version: 5.6

```

89.         //the reading will not be obtained until the function is called a
        second time and the conversion in the ADC has finished
90.         ReadObjectTemperature();// will update every 500ms normally, but will
        skip a cycle when ReadAmbientTemp is using the ADC, making the next update take
        1000ms instead
91.
92.         if (programCounter % 21 == 1 && glassPresent == false)//only check the
        ambient temperature every 5.25 seconds and when class is not present, this is to
        get more accuracy
93.         {
94.             ReadAmbientTemperature();
95.         }
96.
97.         if (programCounter % 2 == 1)//do every 500ms
98.         {
99.             //check if the temperature is above the safe limit of 185
100.            if (currentAmbiantTemp >= 180)
101.            {
102.                //overwrite the display with a warning code
103.                display.WarningBlink(9999);
104.
105.                //turn the buzzer alarm on and off
106.                if (programCounter % 6 == 1)
107.                {
108.                    //turn on the buzzer alarm
109.                    buzz_value = GpioPinValue.Low;
110.                    buzz.Write(buzz_value);
111.                }
112.                else if (programCounter % 12 == 1)
113.                {
114.                    //turn off the buzzer alarm
115.                    buzz_value = GpioPinValue.High;
116.                    buzz.Write(buzz_value);
117.                }
118.
119.                //record the warning in the log every 5 seconds
120.                if (programCounter % 20 == 1)
121.                {
122.                    logManager.AddLogEntry(ERROR_LOG, "HIGH TEMP
WARNING\t" + currentAmbiantTemp.ToString());
123.                    logManager.AddLogEntry(SYSTEM_LOG, "HIGH TEMP
WARNING\t" + currentAmbiantTemp.ToString());
124.                }
125.            }
126.            else
127.            {
128.                //turn off the buzzer alarm
129.                buzz_value = GpioPinValue.High;
130.                buzz.Write(buzz_value);
131.            }

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 26 of 45
		Document No.:1	Version: 5.6

```

132.         }
133.
134.         //save the temperature of the glass when it enters the IR
        sensors vision
135.         if (glassPresent == true && prevGlassPresent == false &&
        newData == true)
136.         {
137.             glassInitialTemp = currentObjectTemp;
138.             display.AdjustBrightness_0_To_8(8);
139.         }
140.
141.         //log the temperature data when the glass leaves the IR
        sensor's vision
142.         if(glassPresent == false && prevGlassPresent == true && newData
        == true)
143.         {
144.             newData = false; //note that the current measurement has
            been recorded to the log
145.
146.             display.AdjustBrightness_0_To_8(0);
147.
148.             logManager.AddLogEntry(DATA_LOG,
            glassInitialTemp.ToString() + "\t" + currentObjectTemp.ToString());
149.         }
150.
151.     }
152.
153.     //watches for sudden large changes in the temperature readings of
    the IR sensor
154.     public void DetectGlassPresent(double currentTemperature)
155.     {
156.         int index1 = 0;
157.
158.         for (index1 = 0; index1 < 4; index1++)
159.         {
160.             double diffTemperature = currentTemperature -
            pastTemps[index1]; //calculate the difference between current and past temps
161.             if(diffTemperature > 500) //glass entering view of IR sensor
162.             {
163.                 prevGlassPresent = glassPresent; //save the previous
                value for detecting a rising or falling edge (glass coming or going)
164.                 glassPresent = true;
165.                 newData = true;
166.                 break; //exit the for loop
167.             }
168.             else if(diffTemperature < 500) //glass exiting view of IR
            sensor
169.             {
170.                 prevGlassPresent = glassPresent; //save the previous
                value for detecting a rising or falling edge (glass coming or going)

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 27 of 45
		Document No.:1	Version: 5.6

```

171.             glassPresent = false;
172.             newData = true;
173.             break;//exit the for loop
174.         }
175.     }
176.
177.     //shift the values in the past temps array
178.     for (index1 = 0; index1 < 3; index1++)
179.     {
180.         pastTemps[index1 + 1] = pastTemps[0];
181.     }
182.     pastTemps[0] = currentTemperature;//put the latest temp into
the register
183. }
184.
185.     public async void ReadObjectTemperature()
186.     {
187.         if (ADC.ADC_initDone == true)
188.         {
189.             if (ADC.conversionRequestedA0 == false &&
ADC.conversionRequestedA1 == false && ADC.performingConversion == false)
190.             {
191.                 //start reading A0 to get object voltage
192.                 await ADC.StartConversion(0);
193.             }
194.
195.             //wait until the conversion is finished
196.             if (ADC.conversionRequestedA0 == true &&
ADC.conversionRequestedA1 == false && ADC.performingConversion == false)
197.             {
198.                 double voltage = ADC.voltage;
199.                 //voltage has been recieved
200.                 ADC.conversionRequestedA0 = false;
201.
202.                 //scale the voltage reading to a temperature
203.                 currentObjectTemp = (((Math.Abs(voltage-.003)) / 5) *
1980) + 212;//212 to 2192 F
204.
205.                 Debug.WriteLine($"VoltObject : {voltage}   tempObject :
{currentObjectTemp:f2}");
206.
207.                 //update the number shown on the display
208.                 display.PrintNumber(Math.Round(currentObjectTemp));
209.
210.                 //look for sudden temperature changes
211.                 DetectGlassPresent(currentObjectTemp);
212.             }
213.         }
214.     }
215.

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 28 of 45
		Document No.:1	Version: 5.6

```

216.         public async void ReadAmbientTemperature()
217.         {
218.             if (ADC.ADC_initDone == true)
219.             {
220.                 if (ADC.conversionRequestedA0 == false &&
ADC.conversionRequestedA1 == false && ADC.performingConversion == false)
221.                 {
222.                     //start reading A0 to get object voltage
223.                     await ADC.StartConversion(1);
224.                 }
225.
226.                 //wait until the conversion is finished
227.                 if (ADC.conversionRequestedA1 == true &&
ADC.conversionRequestedA0 == false && ADC.performingConversion == false)
228.                 {
229.                     double voltage = ADC.voltage;
230.                     //voltage has been recieved
231.                     ADC.conversionRequestedA1 = false;
232.
233.                     //scale the voltage reading to a temperature
234.                     currentAmbiantTemp = (((Math.Abs(voltage)) / 5) + -
0.0132) * 360); //-4 to 356 F
235.
236.                     Debug.WriteLine($"VoltAmb      : {voltage}   tempAmb      :
{currentAmbiantTemp:f2}");
237.                 }
238.             }
239.         }
240.     }
241. }

```

MAX7219_LED_Driver.cs

```

using System;
using System.Threading.Tasks;
using Windows.Devices.Gpio;
//using System.Diagnostics;

/// <summary>
/// Driver for controlling the MAX7219 8 digit LED display using a Raspbery Pi through serial
communication
///David Shaw 2018
/// </summary>

namespace MAX7219_LED_Driver
{
    public sealed class MAX7219_Driver
    {
        // MAX7219 registers
        private const uint REG_DECODEMODE = 0x09;

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 29 of 45
		Document No.:1	Version: 5.6

```

private const uint REG_INTENSITY = 0x0a;
private const uint REG_SCANLIMIT = 0x0b;
private const uint REG_SHUTDOWN = 0x0c;
private const uint REG_DISPTTEST = 0x0f;

//pin assignments
private const int DIN = 5;//pin 29
private GpioPin din;
private GpioPinValue din_value;
private const int CS = 6;//pin 31
private GpioPin cs;
private GpioPinValue cs_value;
private const int CLK = 16;//pin 36
private GpioPin clk;
private GpioPinValue clk_value;

public void InitDisplay()
{
    //creat a new instance of the gpio controller
    var gpio = GpioController.GetDefault();

    //initialize the pins for the display
    din = gpio.OpenPin(DIN);
    cs = gpio.OpenPin(CS);
    clk = gpio.OpenPin(CLK);
    din.SetDriveMode(GpioPinDriveMode.Output);
    cs.SetDriveMode(GpioPinDriveMode.Output);
    clk.SetDriveMode(GpioPinDriveMode.Output);

    /*initialize the display hardware*/
    SetScanLimit_0_To_7(0x07);//turn the whole display on by default
    Task.Delay(100);//wait 100 millisecond

    SetRegister(REG_DECODEMODE, 0xFF);//full decode mode BCD by default
    Task.Delay(100);//wait 100 millisecond

    SetRegister(REG_SHUTDOWN, 0x07);//not in shutdown mode
    Task.Delay(100);//wait 100 millisecond

    SetRegister(REG_DISPTTEST, 0x00);//no display test
    Task.Delay(100);//wait 100 millisecond

    AdjustBrightness_0_To_8(0);//set the brightness to the minimum as the default

    ZeroAll();//zero the display
}

private void WriteByte(uint data)
{
    int index = 8;
    uint mask;//create a mask that will have one bit high at a time
    uint dser;

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 30 of 45
		Document No.:1	Version: 5.6

```

while (index > 0)//loop through the 8 bits of the byte
{
    mask = 0x01;
    mask <= index - 1;//shift to the next bit in the mask

    clk_value = GpioPinValue.Low;
    clk.Write(clk_value);//tick

    dser = data & mask;//set dser to the masked data

    if(dser > 0)
    {
        din_value = GpioPinValue.High;
        din.Write(din_value);
    }
    else
    {
        din_value = GpioPinValue.Low;
        din.Write(din_value);
    }

    clk_value = GpioPinValue.High;
    clk.Write(clk_value);//tick
    index--;
}
}

private void SetRegister(uint reg, uint value)
{
    cs_value = GpioPinValue.Low;
    cs.Write(cs_value);

    WriteByte(reg);//select the register
    Task.Delay(10);//wait 10 millisecond for the hardware to update
    WriteByte(value);//send the data

    cs_value = GpioPinValue.Low;
    cs.Write(cs_value);
    Task.Delay(10);//wait 10 millisecond for the hardware to update
    cs_value = GpioPinValue.High;
    cs.Write(cs_value);
}

public void ZeroAll()//set all characters of the display to zero
{
    uint digits = 1;
    while (digits < 9)
    {
        SetRegister(digits, 0);
        digits++;
    }
}

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 31 of 45
		Document No.:1	Version: 5.6

```

    public void PrintNumber(double c)//convert and print an integer number in format up to
four characters to the display
    {
        double b = c * 10;//this is done to move the first decimal digit left of the point
        uint mask = 0x7F;//mask for ensuring decimal points are off

        uint fourth = Convert.ToUInt32(string.Format("{0}", (int)c / 1000), 16);
        uint third = Convert.ToUInt32(string.Format("{0}", (int)c / 100), 16);
        uint second = Convert.ToUInt32(string.Format("{0}", (int)(c / 10)), 16);
        uint first = Convert.ToUInt32(string.Format("{0}", (int)c), 16);
        uint decimal1 = Convert.ToUInt32(string.Format("{0}", (int)b), 16);
        //output each digit
        //SetRegister(8, 7);
        SetRegister(4, fourth & mask);
        SetRegister(3, third & mask);
        SetRegister(2, second & mask);
        SetRegister(1, first & mask);
        //SetRegister(1, decimal1);
    }

    //adjust the brightness of the display
    public void AdjustBrightness_0_To_8(uint level)
    {
        SetRegister(REG_INTENSITY, level);//set brightness
    }

    //set the scan limit to adjust number of digits turned on
    public void SetScanLimit_0_To_7(uint mask)
    {
        SetRegister(REG_SCANLIMIT, mask);
    }

    //loop from 0 to 9999 to test the display and driver code
    public void DisplayTest()
    {
        int counter = 0;

        for (counter = 0; counter < 9999; counter++)
        {
            Task.Delay(10);
            PrintNumber(counter);
        }
    }

    //blink the brightness and show a number code to show a warning
    public async void WarningBlink(int numberCode)
    {
        AdjustBrightness_0_To_8(0);
        PrintNumber(0);
        await Task.Delay(100);

        AdjustBrightness_0_To_8(8);
        PrintNumber(numberCode);
    }

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 32 of 45
		Document No.:1	Version: 5.6

```

        await Task.Delay(100);
    }
}
}

```

TextFileLogManager.cs

```

using System;
using Windows.Storage; //for working with files in UWP
//using System.Diagnostics;

/// <summary>
/// Class for creating and appending new data to a .txt file log. Each new log entry is placed
/// onto a new line and data within each line is seperated by tabs
/// David Shaw 2018
/// </summary>

namespace TextFileLogManager
{
    class TextFileLog
    {
        //find the apps storage folder
        private StorageFolder storageFolder;
        private StorageFile storageFile;

        public async void InitializeLog()
        {
        }

        public async void AddLogEntry(string logName, string textEntry) //Add a new line of
data to a file in the app's local storage folder
        {
            //find the apps storage folder
            storageFolder = ApplicationData.Current.LocalFolder;

            //open the log file or create a new file if it doesn't exist
            storageFile = await storageFolder.CreateFileAsync(logName,
CreationCollisionOption.OpenIfExists);
            //add the new text on a new line
            await FileIO.AppendTextAsync(storageFile, "\n" + DateTime.Now.ToString("dd/MM/yyyy
h:mm:ss tt") + "\t" + textEntry + "\t");
        }
    }
}

```

CTL_IR_Sensor_Diver.cs

(not used in final version, replaced by ADC)

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 33 of 45
		Document No.:1	Version: 5.6

```

using System;
using System.Linq;
using Windows.Devices.Enumeration;
using Windows.Devices.SerialCommunication;
using Windows.Storage.Streams;
using System.Threading.Tasks;
using System.Diagnostics;//for debugging

/// <summary>
/// Driver for interacting with the CTL IR temperature sensor
/// David Shaw 2018
/// </summary>

```

```

namespace CTL_IR_Sensor_Driver
{
    class IR_Sensor_Interface
    {
        /*----CTL IR sensor commands----*/
        //Basic Functions
        private const byte READ_TEMP_PROCESS = 0x01;
        private const byte READ_TEMP_HEAD = 0x02;
        private const byte READ_TEMP_BOX = 0x03;
        private const byte READ_TEMP_ACT = 0x81;
        //IR Settings
        private const byte READ_EPSILON = 0x04;
        private const byte SET_EPSILON = 0x84;
        private const byte READ_TRANSMISSION = 0x05;
        private const byte SET_TRANSMISSION = 0x85;
        //Aiming
        private const byte READ_SPOT_ILLUMINATION = 0x25;
        private const byte SET_SPOT_ILLUMINATION = 0xA5;
        //Signal Processing
        //Averaging
        private const byte READ_AVG_TIME = 0x06;
        private const byte SET_AVG_TIME = 0x86;
        private const byte READ_AVG_MODE = 0x1C;
    }
}

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 34 of 45
		Document No.:1	Version: 5.6

```

private const byte SET_AVG_MODE = 0x9C;
//Hold functions
private const byte READ_PEAK_HOLD_TIME = 0x08;
private const byte SET_PEAK_HOLD_TIME = 0x25;
private const byte READ_VALLEY_HOLD_TIME = 0x07;
private const byte SET_VALLEY_HOLD_TIME = 0x87;
private const byte READ_ADVANCED_HOLD_MODE = 0x1D;
private const byte SET_ADVANCED_HOLD_MODE = 0x9D;
private const byte READ_ADVANCED_HOLD_THRESHOLD = 0x1E;
private const byte SET_ADVANCED_HOLD_THRESHOLD = 0x9E;
private const byte READ_ADV_HOLD_HYSTERESE = 0x22;
private const byte SET_ADV_HOLD_HYSTERESE = 0xA2;
private const byte READ_PICK_MODE = 0x41;
private const byte SET_PICK_MODE = 0xAE;
//Analog output settings
private const byte READ_ALARMx_MODE = 0x28;
private const byte SET_ALARMx_MODE = 0xA8;
private const byte READ_LOW_END_FOR_OUTPUTS = 0x18;
private const byte SET_LOW_END_FOR_OUTPUTS = 0x98;
private const byte READ_HIGH_END_FOR_OUTPUTS = 0x19;
private const byte SET_HIGH_END_FOR_OUTPUTS = 0x99;
private const byte READ_SKAL_OUT_MIN = 0x11;
private const byte SET_SKAL_OUT_MIN = 0x91;
private const byte READ_SKAL_OUT_MAX = 0x12;
private const byte SET_SKAL_OUT_MAX = 0x92;
//Alarm Settings
private const byte READ_AL1_VALUE = 0x0A;
private const byte SET_AL1_VALUE = 0x8A;
private const byte READ_AL2_VALUE = 0x0B;
private const byte SET_AL2_VALUE = 0x8B;
private const byte READ_AL3_VALUE = 0x0C;
private const byte SET_AL3_VALUE = 0x8C;
private const byte READ_AL4_VALUE = 0x0D;
private const byte SET_AL4_VALUE = 0x8D;
/*Advanced Settings*/
//Sensor Information / Calibration

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 35 of 45
		Document No.:1	Version: 5.6

```

private const byte READ_SERIAL_NUMBER = 0x0E;
private const byte READ_FW_REV = 0x0F;
private const byte READ_SENSOR_INFORMATION = 0x45;
private const byte READ_HEAD_CODE = 0x24;
private const byte SET_HEAD_CODE = 0xA4;
private const byte READ_TWEAK_OFFSET = 0x26;
private const byte SET_TWEAK_OFFSET = 0xA6;
private const byte READ_TWEAK_GAIN = 0x27;
private const byte SET_TWEAK_GAIN = 0xA7;
//Advanced IR-Settings
private const byte READ_AMB_TEMP_SOURCE = 0x13;
private const byte SET_AMB_TEMP_SOURCE = 0x93;
private const byte READ_AMB_TEMP_FIX_VALUE = 0x14;
private const byte SET_AMB_TEMP_FIX_VALUE = 0x94;
private const byte READ_EPS_SOURCE = 0x15;
private const byte SET_EPS_SOURCE = 0x95;
//Advanced Digital Communication Settings
private const byte READ_IF_CHECK_SUM_EXPECTED = 0x2D;
private const byte SET_IF_CHECK_SUM_EXPECTED = 0xAD;
private const byte READ_OUT_BURST_STRING = 0x50;
private const byte SET_OUT_BURST_STRING = 0x51;
private const byte SET_BURST_MODE = 0x52;
private const byte SET_BAUDRATE = 0x82;//Parameter byte1 = 0 - 9600, 1 - 19200, 2 - 38400, 3 - 57600, 4 - 115200
//Loop Maintenance
private const byte READ_OUT_VALUE_FOR_IR_DAC_PERCENTAGE = 0x1A;
private const byte SET_IR_DAC_PERCENTAGE = 0x9A;
private const byte READ_OUT_VALUE_FOR_AMB_DAC_PERCENTAGE = 0x1B;
private const byte SET_AMB_DAC_PERCENTAGE = 0x9B;
private const byte RESET_THE_DAC_PERCENTAGE_OUTPUT = 0x8F;
//Emissivity Determination
private const byte SET_EMISSIVITY_DETERMINATION_TARGET_TEMP = 0x9F;
private const byte SET_EMISSIVITY_DETERMINATION_ACTUAL_TEMP = 0xA0;
private const byte SET_EMISSIVITY_DETERMINATION_STATUS = 0xA1;
//Further Advanced Settings
private const byte SET_DEFAULT = 0xA9;
private const byte READ_PANEL_LOCK = 0x43;

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 36 of 45
		Document No.:1	Version: 5.6

```

private const byte SET_PANEL_LOCK = 0x44;
private const byte READ_TEMP_UNIT = 0x09;
private const byte SET_TEMP_UNIT = 0x89;

//create serial objects
private SerialDevice serialPort = null;
DataReader dataReaderObject = null;
DataWriter dataWriterObject = null;
//private uint BytesReceived { get; }
//private uint data;

//private ObservableCollection<DeviceInformation> listOfDevices;

//pin assignments
//private const int DIN = 5;//pin 29
//private GpioPin din;
//private GpioPinValue din_value;

public void Init_IR_Sensor()
{
    //creat a new instance of the gpio controller
    //var gpio = GpioController.GetDefault();
    //discover the available ports
    //listOfDevices = new ObservableCollection<DeviceInformation>()
    ListAvailablePorts();
}

public double RandomTempTest()
{
    Random r = new Random();//random number for testing
    double rNumber = r.Next(100, 2500);
    //int rNumber2 = r.Next(0, 9);
    return rNumber;(((rNumber*10) + rNumber2) / 10;
}

private async Task<uint> ReadSerialData()

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 37 of 45
		Document No.:1	Version: 5.6

```

{
    uint data = 0;
    string dataString = "";

    dataReaderObject = new DataReader(serialPort.InputStream);
    try
    {
        uint bytesReceived = await dataReaderObject.LoadAsync(128);
        if(bytesReceived > 0)
        {
            //read the data as a string and trim it
            dataString = dataReaderObject.ReadString(bytesReceived).Trim();
            data = dataReaderObject.ReadUInt32();
        }
    }
    catch (Exception ex)
    {
        //error recieving data from input stream
        Debug.WriteLine("Error in ReadSerialData(): " + ex.Message);
        data = 0;
        ListAvailablePorts();
    }
    finally
    {
        if (dataReaderObject != null)
        {
            dataReaderObject.DetachStream();
            dataReaderObject = null;
        }
    }
    return data;
}

public async Task<double> ReadTemperature()
{
    uint temperature = 0;

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 38 of 45
		Document No.:1	Version: 5.6

```

try//send the read command to the sensor then read what the information it sends back
{
    WriteByteToSensor(READ_TEMP_PROCESS);
    temperature = await ReadSerialData();
    Debug.Write("Raw serial returned to ReadTemp(): " + temperature.ToString());
}
catch (Exception ex)
{
    Debug.Write("error reading temperature: " + ex.Message);
    temperature = 1000;
}

//return the temperature as a floating point number
return (temperature - 1000) / 10;
}

//write a byte to the serial port
private async void WriteByteToSensor(byte sensorCommand)
{
    Task<UInt32> asyncTask;

    try
    {
        //if there is a connected serial port then creat a DataWriter and send the chosen command
        if(serialPort != null)
        {
            dataWriterObject = new DataWriter(serialPort.OutputStream);

            //input the byte command into the writer object
            dataWriterObject.WriteByte(sensorCommand);

            //launch an async task to complete the operation
            asyncTask = dataWriterObject.StoreAsync().AsTask();
            //for debugging
            UInt32 bytesWritten = await asyncTask;

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 39 of 45
		Document No.:1	Version: 5.6

```

        if(bytesWritten > 0)
        {
            Debug.Write(sensorCommand.ToString() + "sent");
        }
    }
else
{
    Debug.Write("Serial device not connected");
}
}
catch (Exception ex)
{
    Debug.Write("Exception: " + ex.Message);
}
finally
{
    {
        // cleanup
        if (dataWriterObject != null)
        {
            dataWriterObject.DetachStream();
            dataWriterObject = null;
        }
    }
}

//perform a check sum to ensure that a set command has been properly transmitted
private void CheckSum(string checkmessage)
{
    byte xorTotalByte = 0;
    for (int i = 0; i < checkmessage.Length; i += 2)
    {
        //byte b = byte.Parse(checkmessage.Substring(i, 2), NumberStyles.AllowHexSpecifier);
        //xorTotalByte ^= b;
    }
}

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 40 of 45
		Document No.:1	Version: 5.6

```

//find all the available ports on the device
private async void ListAvailablePorts()
{
    try
    {
        string aqs = SerialDevice.GetDeviceSelector();
        var dis = await DeviceInformation.FindAllAsync(aqs);

        var selectedPort = dis.First();//select the first port
        serialPort = await SerialDevice.FromIdAsync(selectedPort.Id);

        Debug.Write(selectedPort.Id.ToString());

        //default settings for the CTL IR sensor
        serialPort.BaudRate = 9600;
        serialPort.DataBits = 8;
        serialPort.Parity = SerialParity.None;
        serialPort.StopBits = SerialStopBitCount.One;
        //timeout after one second
        serialPort.ReadTimeout = TimeSpan.FromMilliseconds(1000);
    }
    catch (Exception ex)
    {
        //add a warning or notification here
        Debug.Write("Error in ListAvailablePorts(): " + ex.Message);
    }
}

//check for high temp alarms in the sensor and return a number code
public async Task<uint> CheckForHighTempAlarm()
{
    uint warningCode = await ReadSerialData();//place holder
    return 0;
}
}

```


Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 41 of 45
		Document No.:1	Version: 5.6

VBA macro code module running in Excel

```

Sub ImportTextFile()
    Dim fPath As String
    Dim continue As Boolean
    Dim lastRow As Integer
    Dim wb As Excel.Workbook
    Dim ws As Excel.Worksheet

    Set wb = Excel.ActiveWorkbook
    Set ws = Excel.ActiveSheet

    Application.ScreenUpdating = False

    fPath = "C:\Users\David\Desktop\test.txt"
    Sheets("Settings").Select
    fPath = CStr(Cells(2, 3))

    'check that the file exists
    'continue = fileExists(fPath, True)
    If (Dir(fPath) <> "") Then

        'the fieldinfo array needs to be extended to match your number of columns
        Workbooks.OpenText FileName:=fPath, Origin:=xlMSDOS, StartRow:=1, DataType:=xlDelimited,
        TextQualifier:=xlTextQualifierNone, Comma:=False, Space:=True, FieldInfo:=Array(Array(1, xlTextFormat), Array(2,
        xlTextFormat), Array(3, xlTextFormat))

        'move the data into the main Workbook
        Sheets(1).Move Before:=wb.Sheets(1)

        'find the end of the data then copy all of it
        lastRow = Cells(Rows.Count, 1).End(xlUp).Row
        Range(Cells(1, 1), Cells(lastRow, 3)).Copy

        Sheets("Sheet Glass Tracking").Select

        'get the last row of the main workbook then paste there
        lastRow = Cells(Rows.Count, 1).End(xlUp).Row

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 42 of 45
		Document No.:1	Version: 5.6

```
Range(Cells(lastRow + 1, 1), Cells(lastRow + 1, 1)).Select
ActiveSheet.Paste
```

```
Application.DisplayAlerts = False
Worksheets(1).Delete
Application.DisplayAlerts = True
```

```
Sheets("Sheet Glass Tracking").Select
Else
```

```
'manually select the file if it can't be found
fPath = SelectFiles(False)
```

```
If (fPath <> "") Then
    Sheets("Settings").Cells(2, 3) = fPath
    Call ImportTextFile
End If
End If
End Sub
```

From VBA macro code library written by me
Subroutine used by the module in code appendix 6.

```
Public Function SelectFiles(Optional ByVal openToThisWrkBkPath As Boolean = True) As String 'let the user select a file
and return the files full name
```

```
    Dim fd As Office.FileDialog
```

```
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
```

```
    If (openToThisWrkBkPath) Then
        fd.InitialFileName = Application.ActiveWorkbook.Path 'open the current file path
    End If
```

```
    With fd
```

```
        .AllowMultiSelect = False
```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 43 of 45
		Document No.:1	Version: 5.6

```
' Set the title of the dialog box.
.Title = "Please select the file."
```

```
' Clear out the current filters, and add our own.
.Filters.Clear
.Filters.Add "Excel", "*.xls"
.Filters.Add "All Files", "*.*"
.Filters.Add "Text", "*.txt"
```

```
' Show the dialog box. If the .Show method returns True, the
' user picked at least one file. If the .Show method returns
' False, the user clicked Cancel.
If .Show = True Then
    txtFileName = .SelectedItems(1)
End If
```

```
End With
```

```
SelectFiles = txtFileName 'return the file name
```

```
End Function
```

ADC_I2C.cs

```
using System;
using System.Threading.Tasks;
using Windows.Devices.I2c;
using System.Diagnostics;
using Windows.Devices.Enumeration;
using Windows.Devices.Gpio;
using TextFileLogManager;

/// <summary>
///
/// Connect to the ADS1115 ADC and read the values of analog input pins A0 and A1. Then
convert and scale the voltages to temperatures.
/// David Shaw 2019
///
/// </summary>
```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 44 of 45
		Document No.:1	Version: 5.6

```

namespace ADC
{
    class ADC_I2C
    {
        private I2cDevice converter;
        private int READY_PIN = 27; //pin13
        private GpioPin GpioInputPin;

        public bool ADC_initDone = false;
        public bool conversionRequestedA0 = false;
        public bool conversionRequestedA1 = false;
        public bool performingConversion = false;
        public double voltage = 0;
        public int lastConversionForPin = 1;

        private async void ReadConversionResultVoltage(GpioPin sender,
GpioPinValueChangedEventArgs args)
        {
            // Read conversion register to get result
            var bytearray = new byte[2];
            converter.WriteRead(new byte[] { 0x0 }, bytearray);

            // Convert byte array to an int16
            if (BitConverter.IsLittleEndian)
            {
                Array.Reverse(bytearray);
            }
            var value = BitConverter.ToInt16(bytearray, 0);

            //calculate voltage based on ADC parameters
            // formula is voltage = (value * FSR) / 32767.0
            voltage = ((value * 6.144) / 32767.0); //0 to 5 V

            //the conversion is finished
            performingConversion = false;
        }

        public async Task StartConversion(int analogInput)
        {
            if (ADC_initDone == true)
            {
                switch (analogInput)
                {
                    case 0:
                        conversionRequestedA0 = true;
                        // Write in the config register. 0xc5 0Xe0 = 1 100 000 1 111
00000
                        // listen to A0, FSR of 6.144V, single-shot conversion mode, 860
SPS, assert after one conversion (= READY signal)
                        converter.Write(new byte[] { 0x01, 0xc1, 0xe0 });

```

Issued By: David Shaw	Approved By: Professor Elaine Cooney	Effective Date: February 28, 2019	Page 45 of 45
		Document No.:1	Version: 5.6

```

        performingConversion = true;
        lastConversionForPin = 0;
        break;
    case 1:
        conversionRequestedA1 = true;
        // Write in the config register. 0xd5 0Xe0 = 1 101 000 1 111
00000        // listen to A1, FSR of 6.144V, single-shot conversion mode, 860
SPS, assert after one conversion (= READY signal)
        converter.Write(new byte[] { 0x01, 0xd1, 0xe0 });
        performingConversion = true;
        lastConversionForPin = 1;
        break;
    default:
        conversionRequestedA0 = false;
        conversionRequestedA1 = false;
        break;
    }
}

}

public async Task InitializeAsync()// Initialize the ADC
{
    ADC_initDone = false;

    //Setup I2c communication
    var i2cSettings = new I2cConnectionSettings(0x48)
    {
        BusSpeed = I2cBusSpeed.FastMode,
        SharingMode = I2cSharingMode.Shared
    };
    //find and connected to I2c device
    var i2c1 = I2cDevice.GetDeviceSelector("I2C1");
    var devices = await DeviceInformation.FindAllAsync(i2c1);
    converter = await I2cDevice.FromIdAsync(devices[0].Id, i2cSettings);

    // Configure the Lo_thresh (0x02) and Hi_Thresh (0x03) registers so the READY
signal will be sent
    converter.Write(new byte[] { 0x02, 0x00, 0x00 });
    converter.Write(new byte[] { 0x03, 0xff, 0xff });

    //Set the READY pin on the Pi and tie it to InGpioPinOnValueChanged
    var gpio = GpioController.GetDefault();
    GpioInputPin = gpio.OpenPin(READY_PIN);
    GpioInputPin.ValueChanged += ReadConversionResultVoltage;

    ADC_initDone = true;
}
}
}

```